BLOG

Understanding the PostgreSQL Architecture

Chitij Chauhan

Published: October 2, 2017

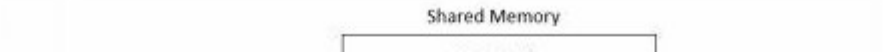
Last Updated: April 11, 2025

PostgreSQL is probably the most advanced database in the open source relational database market. It was first released in 1989, and since then, there have been a lot of enhancements. According to db-engines, it is the fourth most used database at the time of writing.

In this blog, we will discuss PostgreSQL internals, its architecture, and how the various components of PostgreSQL interact with one another. This will serve as a primer for anyone interested in operationalizing a PostgreSQL database in a production environment.

PostgreSQL Architecture

The physical structure of PostgreSQL is very simple. It consists of shared memory and a few background processes and data files. (See Figure 1-1)



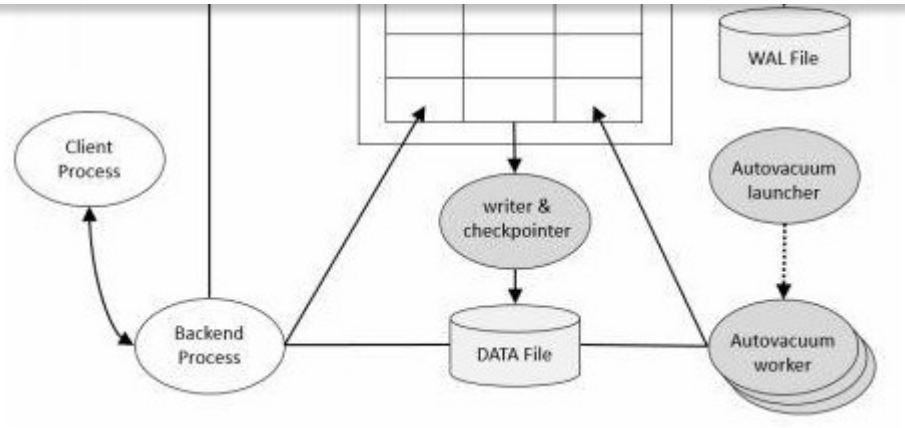


Figure 1-1. PostgreSQL structure

Shared Memory

Shared Memory refers to the memory reserved for database caching and transaction log caching. The most important elements in shared memory are Shared Buffer and WAL buffers

Shared Buffer

The purpose of Shared Buffer is to minimize DISK IO. For this purpose, the following principles must be met

- You need to access very large (tens, hundreds of gigabytes) buffers quickly.
- You should minimize contention when many users access it at the same time.
- Frequently used blocks must be in the buffer for as long as possible

WAL Buffer

The WAL buffer is a buffer that temporarily stores changes to the database. The contents stored in the WAL buffer are written to the WAL file at a predetermined point in time. From a backup and recovery point of view, WAL buffers and WAL files are very important.

PostgreSQL Process Types

PostgreSQL has four process types.

1. Postmaster (Daemon) Process
2. Background Process

Postmaster Process

The Postmaster process is the first process started when you start PostgreSQL. At startup, performs recovery, initialize shared memory, and run background processes. It also creates a backend process when there is a connection request from the client process. (See Figure 1-2)

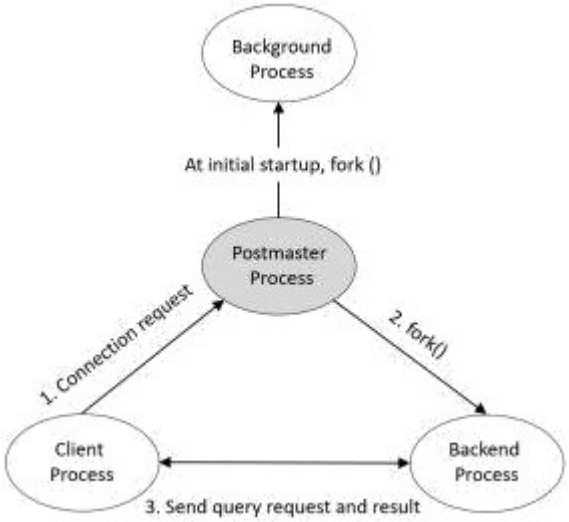


Figure 1-2. Process relationship diagram

If you check the relationships between processes with the pstree command, you can see that the Postmaster process is the parent process of all processes. (For clarity, I added the process name and argument after the process ID)

```
$ pstree -p 1125
postgres(1125) /usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data
        ├─postgres(1249) postgres: logger process
        ├─postgres(1478) postgres: checkpointer process
        ├─postgres(1479) postgres: writer process
        ├─postgres(1480) postgres: wal writer process
        ├─postgres(1481) postgres: autovacuum launcher process
        ├─postgres(1482) postgres: archiver process
        ├─postgres(1483) postgres: stats collector process
```

Background Process

The list of background processes required for PostgreSQL operation are as follows. (See Table 1-1)

logger	Write the error message to the log file.
checkpointer	When a checkpoint occurs, the dirty buffer is written to the file.
writer	Periodically writes the dirty buffer to a file.
wal writer	Write the WAL buffer to the WAL file.
Autovacuum launcher	Fork autovacuum worker when autovacuum is enabled.It is the responsibility of the autovacuum daemon to carry vacuum operations on bloated tables on demand
archiver	When in Archive.log mode, copy the WAL file to the specified directory.
stats collector	DBMS usage statistics such as session execution information (pg_stat_activity) and table usage statistical information (pg_stat_all_tables) are collected.

Backend Process

The maximum number of backend processes is set by the max_connections parameter, and the default value is 100. The backend process performs the query request of the user process and then transmits the result. Some memory structures are required for query execution, which is called local memory. The main parameters associated with local memory are:

1. work_mem Space used for sorting, bitmap operations, hash joins, and merge joins. The default setting is 4 MB.
2. Maintenance_work_mem Space used for Vacuum and CREATE INDEX . The default setting is 64 MB.
3. Temp_buffers Space used for temporary tables. The default setting is 8 MB.

Client Process

Client Process refers to the background process that is assigned for every backend user connection.Usually the postmaster process will fork a child process that is dedicated to

Database Structure

Here are some things that are important to know when attempting to understand the database structure of PostgreSQL.

Items related to the database

1. PostgreSQL consists of several databases. This is called a database cluster.
2. When initdb () is executed, template0 , template1 , and postgres databases are created.
3. The template0 and template1 databases are template databases for user database creation and contain the system catalog tables.
4. The list of tables in the template0 and template1 databases is the same immediately after initdb (). However, the template1 database can create objects that the user needs.
5. The user database is created by cloning the template1 database.

Items related to the tablespace

1. The pg_default and pg_global tablespaces are created immediately after initdb().
2. If you do not specify a tablespace at the time of table creation, it is stored in the pg_dafault tablespace.
3. Tables managed at the database cluster level are stored in the pg_global tablespace.
4. The physical location of the pg_default tablespace is $PGDATAbase.
5. The physical location of the pg_global tablespace is $PGDATAglobal.
6. One tablespace can be used by multiple databases. At this time, a database-specific subdirectory is created in the table space directory.
7. Creating a user tablespace creates a symbolic link to the user tablespace in the $PGDATAtblspc directory.

Items related to the table

1. There are three files per table.
2. One is a file for storing table data. The file name is the OID of the table.
3. One is a file to manage table free space. The file name is OID_fsm .
4. One is a file for managing the visibility of the table block. The file name is OID_vm .
5. The index does not have a _vm file. That is, OID and OID_fsm are composed of two

Other Things to Remember...

The file name at the time of table and index creation is OID, and OID and pg_class.relfilenode are the same at this point. However, when a rewrite operation (Truncate , CLUSTER , Vacuum Full , REINDEX , etc.) is performed, the relfilenode value of the affected object is changed, and the file name is also changed to the relfilenode value. You can easily check the file location and name by using pg_relation_filepath ('< object name >'). template0, template1, postgres database

Running Tests

If you query the pg_database view after initdb() , you can see that the template0 , template1 , and postgres databases have been created.

```
select oid, datname, datistemplate, datallowconn
from   pg_database order by 1;
  oid | datname | datistemplate | datallowconn
------- _          +--------------- +--------------
    1 | template1  | t           | t
13321 | template0 | t             | f
13322 | postgres  | f             | t
```

- Through the datistemplate column, you can see that the template0 and template1 databases are database for template for user database creation.
- The datlowconn column indicates whether the database can be accessed. Since the template0 database can't be accessed, the contents of the database can't be changed either.
- The reason for providing two databases for the templateis that the template0 database is the initial state template and the template1 database is the template added by the user.
- The postgres database is the default database created using the template1 database. If you do not specify a database at connection time, you will be connected to the postgres database.
- The database is located under the $PGDATA/base directory. The directory name is the database OID number.

```
[postgres@pgserver data]$ ls -l $PGDATA/base
drwx------ . 2 postgres postgres 8192 Nov 4 19:34 1
```

Create User Database

The user database is created by cloning the template1 database. To verify this, create a user table T1 in the template1 database. After creating the mydb01 database, check that the T1 table exists. (See Figure 1-3.)

```
-- Create a T1 table in the template database.
template1=# create table t1 (c1 integer);

-- Create the mydb01 database.
postgres=# create database mydb01;

-- After connecting to mydb01 database, check if T1 table
exists. mydb01=# \d t1
     Table "public.t1"
Column  | Type     | Modifiers
--------+---------  +-----------
c1   | integer |
```

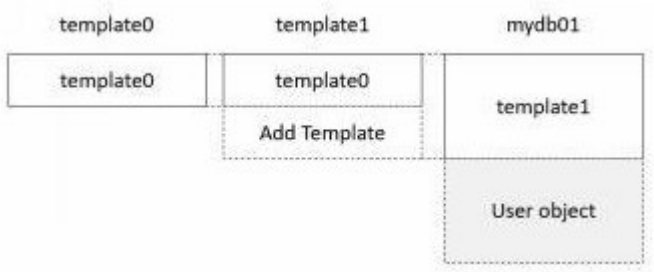


Figure 1-3. Relationship between Template Database and User Database

pg_default tablespace

If you query pg_tablespace after initdb (), you can see that the pg_default and pg_global tablespaces have been created.

```
postgres=# select oid, * from pg_tablespace;
oid | spcname     | spcowner | spcacl | spcoptions
------+-----------  +---------- +---------+------------
1663 | pg_default |    10 |      |
1664 | pg_global |         10
```

The location of the pg_default tablespace is $PGDATAbase. There is a subdirectory by database OID in this directory. (See Figure 1-4)

```
drwx------ . 2 postgres postgres 8192 Nov 4 20:02 13321
drwx------ . 2 postgres postgres 8192 Nov  4 20:02 13322
drwx------ . 2 postgres postgres 8192 Nov  4 20:06 16396
```

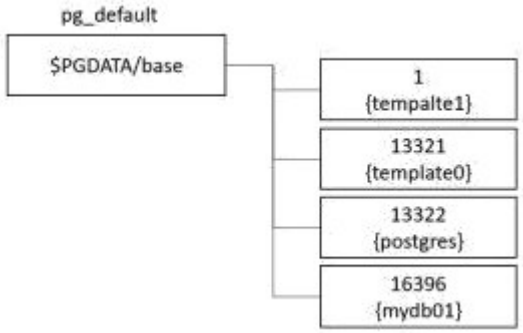


Figure 1-4. Pg_default tablespace and database relationships from a physical configuration perspective

pg_global tablespace

The pg_global tablespace is a tablespace for storing data to be managed at the 'database cluster' level.

- For example, tables of the same type as the pg_database table provide the same information whether they are accessed from any database. (See Figure 1-5)
- The location of the pg_global tablespace is $PGDATAglobal.

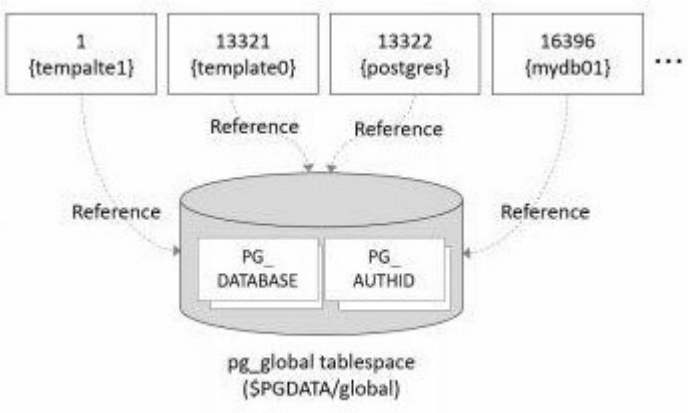


Figure 1-5. Relationship between pg_global tablespace and database

Create User Tablespace

```
postgres=# create tablespace myts01 location '/data01';
```

The pg_tablespace shows that the myts01 tablespace has been created.

```
------+----------+---------+--------+------------
1663|pg_default|    10|      |
1664|pg_global |    10|      |
24576|myts01    |    10|      |
```

Symbolic links in the $PGDATA/pg_tblspc directory point to tablespace directories.

```
[postgres@pgserver ~]$ ls -l $PGDATA/pg_tblspc
lrwxrwxrwx. 1 postgres postgres 7 Nov 8 15:10 24576 -> /data01
```

Connect to the postgres and mydb01 databases and create the table.

```
[postgres@pgserver ~]$ psql -p 5436 -d postgres
-- Create a table.
postgres=# create table t1 (c1 integer) tablespace myts01;

-- Check the OID.
postgres=# select oid from pg_class where relname='t1';
  oid
-------
 24577

[postgres@pgserver ~]$ psql -p 5436 -d mydb01

-- Create a table.
mydb01=# create table t1 (c1 integer) tablespace myts01;

-- Check the OID.
mydb01=# select oid from pg_class where relname='t1';
  oid
-------
 24580
```

If you look up the /data01 directory after creating the table, you will see that the OID directory for the postgres and mydb01 databases has been created and that there is a file in each directory that has the same OID as the T1 table.

```
[postgres@pgserver ~]$ ls -Rl /data01
/data01:
drwx------. 4 postgres postgres 30 Nov 8 15:19 PG_9.6_201608131

/data01/PG_9.6_201608131:
drwx------. 2 postgres postgres 18 Nov 8 15:18 13322
drwx------. 2 postgres postgres 18 Nov 8 15:19 16396

/data01/PG_9.6_201608131/13322:
-rw-------. 1 postgres postgres 40960 Nov 8 15:40 24577
```

How to Change Tablespace Location

PostgreSQL specifies a directory when creating tablespace. Therefore, if the file system where the directory is located is full, the data can no longer be stored. To solve this problem, you can use the volume manager. However, if you can't use the volume manager, you can consider changing the tablespace location. The order of operation is as follows.

```
-- Shutdown PostgreSQL.
/usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data stop

-- Copy the myts01 tablespace to the new file system.
[postgres@pgserver data01]$ cp -rp /data01/PG* /data02

-- Check the contents of the pg_tblspc
directory. [postgres@pgserver pg_tblspc]$ ls -l
lrwxrwxrwx. 1 postgres postgres 7 Nov 8 15:52 24576 -> /data01

-- Delete the symbolic link.
[postgres@pgserver pg_tblspc]$ rm 24576

-- Create a new symbolic link.
[postgres@pgserver pg_tblspc]$ ln -s /data02 24576

-- Confirm the contents
[postgres@pgserver pg_tblspc]$ ls -l
lrwxrwxrwx. 1 postgres postgres 7 Nov 8 15:53 24576 -> /data02

-- Startup PostgreSQL.
/usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data start
```

Note: Tablespaces are also very useful in environments that use partition tables. Because you can use different tablespaces for each partition table, you can more flexibly cope with file system capacity problems.

What is Vacuum?

Vacuum does the following:

1. Gathering table and index statistics
2. Reorganize the table
3. Clean up tables and index dead blocks
4. Frozen by record XID to prevent XID Wraparound

The biggest difference I think is the MVCC model and the existence of a shared pool. This is also considered a feature of PostgreSQL. (See Table 1-2)

Item	ORACLE	PostgreSQL
MVCC model	UNDO	Store previous
Implementation method	Segment	record within block
Shared Pool	exists	it does not exist

Differences in the MVCC Model

To increase concurrency, you must follow the principle that “read operations do not block write operations and write operations should not block read operations”. To implement this principle, a Multi Version Concurrency Control (MVCC) is required. Oracle uses UNDO segments to implement MVCC. On the other hand, PostgreSQL uses a different way to store previous records in a block. It uses the transaction XID and xmin and xmax pseudo columns for transaction row versioning.

Shared Pool

PostgreSQL does not provide a shared pool. This is somewhat embarrassing for users familiar with Oracle. Shared Pool is a very important and essential component in Oracle. PostgreSQL provides the ability to share SQL information at the process level instead of the Shared Pool. In other words, if you execute the same SQL several times in one process, it will hard-parse only once.

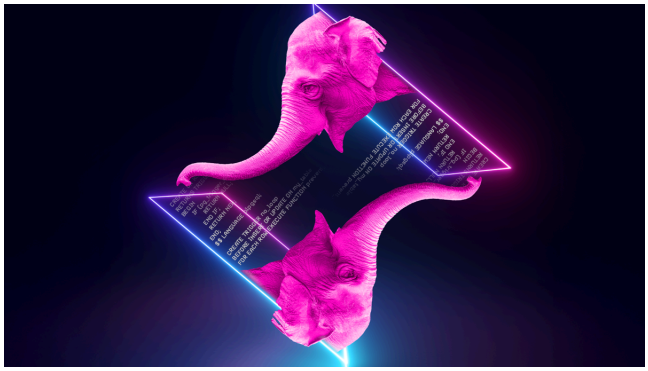
Wrapping up

By understanding the PostgreSQL architecture and concepts, you will have a good grounding from which to inform your use of it in production workloads. But, you don’t have to go it alone — modern tooling, like our very own ClusterControl (which you can try free for 30 days), make deploying and running PostgreSQL clusters a lot simpler in any

automating PostgreSQL operations with ClusterContol in this white paper. In the meantime, don't forget to join our monthly newsletter below and follow us on Twitter or LinkedIn to stay up-to-date with the Postgres and open-source database world.

PostgreSQL Bi-Directional Replication Demystified — From Theory to Operations

Register for the webinar today!



RELATED PRODUCTS

Worry-free database automation

CATEGORY

CONFIGURATION MANAGEMENT POSTGRESQL

Subscribe to our newsletter

You'll get two emails every month full of fresh database ops tips and strategic considerations.

Email

I'm not a robot

reCAPTCHA
Privacy - Terms

Subscribe

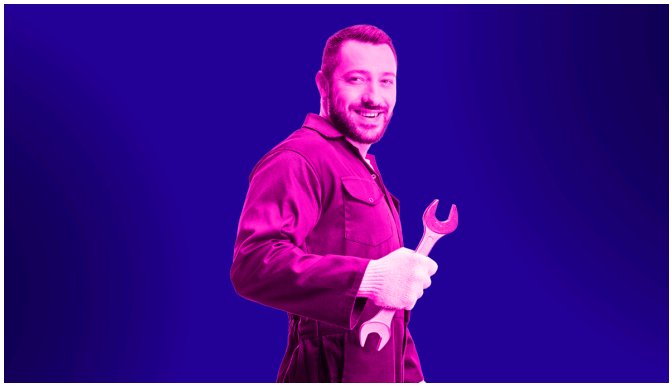
Recommended

Proactive MongoDB Monitoring (Developer Studio/Advisors angle)

August 6, 2021 Agus Syafaat

Performance Tuning for Redis

July 26, 2021 Paul Namuag

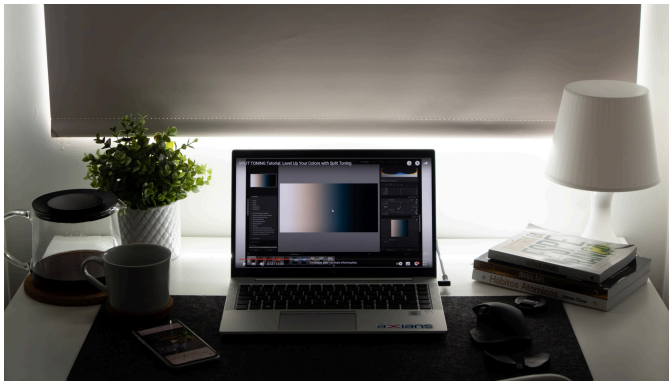


ACL using s9s CLI

July 22, 2021 Zamani Rahman

Installing Redis Cluster (cluster mode enabled) with auto failover

July 6, 2021 Sebastian Insausti



Subscribe

below to be notified of **fresh posts**

Supported databases

ClusterControl

PostgreSQL

MongoDB

TimescaleDB

Elasticsearch

MariaDB

MySQL

MySQL Galera

SQL Server for Linux

Redis

Valkey

CCX

PostgreSQL

MySQL

Redis

MariaDB

SQL Server

Products

ClusterControl

CCX

Solutions

On-premises

Multi cloud

Hybrid cloud

Disaster Recovery

CCX Cloud

CCX for Service Providers

Support
Contact us
Sovereign DBaaS
Compliance
Community
Careers
ClusterControl Training
Partners
ClusterControl
Certification
Brand Book

You'll get two emails every month full of fresh database ops tips and strategic considerations.

Email

I'm not a robot

reCAPTCHA
Privacy - Terms

Subscribe

Terms of ServiceService Level AgreementData Processing AgreementPrivacy Policy

© Copyright 2014-2025 Severalnines AB. All rights reserved.
Severalnines, ClusterControl, and CCX are registered trademarks in the US, UK, and EU. The 3rd-party trademarks on this site are the property of their respective owners and are used for referential purposes only.